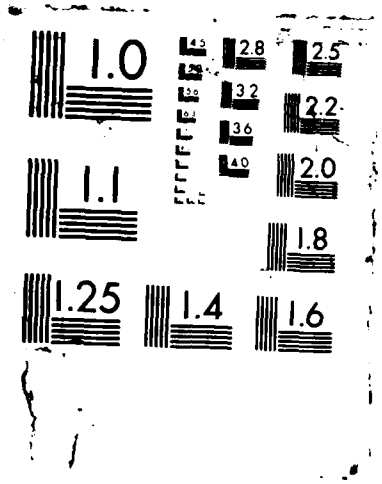
AD-A190 236 TRANSFORMATIONS OF CONCURRENT ALGORITHMS FOR HIGHLY PARALLEL SYSTEMS: A O..(U) INDIANA UNIV AT BLOOMINGTON DEPT OF COMPUTER SCIENCE D GANNON OCT 87 1/1

UNCLASSIFIED AFOSR-TR-87-1737 AFOSR-86-0147 F/G 12/6 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

DTIC FILE COPY (2)

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
AD-A190 236	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 87-1737
6a. NAME OF PERFORMING ORGANIZATION: Indiana University Foundation	6b. OFFICE SYMBOL (If applicable):
6c. ADDRESS (City, State and ZIP Code): Bloomington, Indiana	7a. NAME OF MONITORING ORGANIZATION: Air Force Office of Scientific Research
	7b. ADDRESS (City, State and ZIP Code): Directorate of Mathematical & Information Sciences, Bolling AFB DC 20332-6448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: AFOSR	8b. OFFICE SYMBOL (If applicable): NM
8c. ADDRESS (City, State and ZIP Code): Bolling AFB DC 20332-6448	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: AFOSR-86-0147
	10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
61102F	2304	A3	

11. TITLE (Include Security Classification): Transformations of Concurrent Algorithms for Highly Parallel Systems: A one Year Project Summary Report.

12. PERSONAL AUTHOR(S): Dennis Gannon

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Final	FROM 01 Oct 87 TO	Oct 87	5

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This project has investigated transformations of algorithms for highly concurrent systems. Two technical reports resulting from this research which were completed during the first years effort were entitled "Applying AI techniques to program optimization for parallel computers" and "Strategies for cache and local memory management by global program transformations."

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
Maj. John Thomas	(202) 767-5026	NM

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE. UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

Transformations of Concurrent Algorithms for Highly Parallel Systems: A One Year Project Summary Report.

Dennis Gannon

Dept. of Computer Science Indiana, University Bloomington, Indiana

ABSTRACT

This report describes the activities carried out under AFOSR GRANT 86-0147 covering the period from the starting date to Oct. 1. 1987.

1. INTRODUCTION

It has become a certainty that Multiple Instruction Stream, Multiple Data Stream (MIMD) parallel architectures are going to play a major role in all aspects of high speed computer design for the foreseeable future. What is not clear is whether we will be able to devise a means to design algorithms and software for these machines that transcends our current ad-hoc, nonportable techniques. In this research project we have focused on the portability issue from the perspective of parallel algorithm design and how it effects the internal organization of advanced compilers. The eventual goal of the project is to produce an "expert system" that can help users transform large, complex applications from one highly parallel machine to another. Our basic strategy has been to follow the following plan:

1. Build an experimental research laboratory for parallel computation.

   Parallel computation has both a theoretical foundation and an experimental component. Because we are interested in both the process of porting programs and the behavior of the algorithms on different target machines, it was essential to build a laboratory for parallel computation. With help from the AFOSR and ONR University Instrumentation Program (and nearly $400,000 from the university) we established a parallel computation research lab. The lab houses two machines of great interest to us. One is a 16 Processor BBN Butterfly parallel shared memory computer. The other is an Alliant FX/8 4 CE vector multiprocessor. These two systems provides an outstanding basis for research on the portability problem because they have radically different architectures but are both still classified as shared memory parallel computers.

2. Design an experimental program of research that would shed light on the problems involved with restructuring parallel programs for different machines.

Because the objective is to build tools that "understand" the problems of parallelizing programs for given machines, we needed experience in seeing how algorithms work and how they differ in the organization of the parallelism when optimized for these two machines. In section 2 of this report we will summarize

the algorithms that we worked with and briefly describe some of the results.

3 Attempt to provide a mathematical characterization of the properties of the machines and how algorithms must be restructured to run on them.

So far we have focused on the properties of memory hierarchy such as cache memory and processor local memory. We have developed a mathematical model of how cache behavior can be related to program data dependencies. This work was done in collaboration with William Jalby of INRIA in Paris and Kyle Gallivan of CSRD in Urbana. The results were presented in an invited paper in the first international conference on supercomputing in Athens Greece in July of 1987. A copy of this paper is included as an appendix to this report. The next step is to design a mathematical model of task granularity and synchronization. We are still working on this problem and should be able to report some results by the end of the contract period.

4 Attempt to design a model of machine architecture that can be embedded into the inference engine or knowledge base of an expert system for program restructuring.

Our first attempt at this is nearly complete. Graduate students Ko-Yang Wang has designed a prototype inference system for restructuring programs. The system works with the user who selects the part of the program on which to focus the systems expertise. The system then consults the knowledge base that describes the properties of the target machine and derives a suggested sequence of program transformations that best optimizes the section of code for execution on the target. A preliminary report on the ideas in the system has been written and will be published in a volume edited by Doug DeGroot and Kai Hwang on supercomputing and A.I. machines. A copy of this report is attached as an appendix to this report.

2. ALGORITHM EXPERIMENTS

Our experimental philosophy is to lean the what an expert system for program restructuring should do by becoming experts at porting and restructuring codes for different parallel machines. We have spent 10 years at this with a variety of prototype machines and now one year with our own laboratory facilities. A large number of experiments have been carried out. Because this work has not been published anywhere we thought it would be a good idea to give a brief summary of this activity here.

Our target machines include the two systems that we operate in our lab. One is a 16 processor butterfly shared memory computer and the other is an Alliant FX/8 with 4 vector processors. There are four experiments described here. Two are complete and two are still in progress. All of this work was supported, at least in part, by this grant.

Ray Traced Computer Graphics.

This experiment was carried out to test the problems of extracting the parallelism in an application that is very computationally intensive but also has data structures that are more closely associated with recursive algorithms than the traditional numerical codes. The algorithm works by following optics in reverse. Light rays are "traced" from the eye of the viewer back into the scene

where they reflect off and refract through objects. Because each ray is independent of all of the others the task is completely parallel. One processor can be assigned to each light ray and massive parallelism can be obtained. This was done by a team of students for the butterfly and reasonably good performance resulted.

The primary problem was that the object data base was stored in globally shared memory and each processor needed constant access to this data base. Because a global memory module can only be reference at constant rate and only one processor may have access to the data in that memory module at any given instant of time, there is an upper bound on the number of processors that may share an object that they frequently reference without causing some conflicts and delays. By distributing the shared data through the set of memory memory modules in a uniform manner, we were able to reduce the contention and increase performance.

An important lesson was learned here. For large shared memory parallel systems the distribution of data can, and must, be a major task of any compiler that tries to optimize performance.

The Alliant FX/8 presented a different set of problems. First, the processor on the Alliant machine contains complex and powerful vector hardware. The problem is that it is not easy to exploit on this algorithm. We did, however, discover that there are a number of ways that it can be exploited for simple computations that must be carried out for each ray. For example, each ray must be intersected with each object in the scene (for a simple ray tracing algorithm). This process may be easily vectorized and good performace results. For more complex algorithms this task is not needed. We are still studing the problem of how to provide effective exploitation of vector hardware on the problem.

Numerical FFT algorithms.

Numerical FFTs are just one of many numerical computations we have worked on. In all cases we have found one striking difference between the effective use of the parallel hardware one our two systems. In particular, we have discovered that on the Alliant system the memory hierarchy is such that processors "like" to share common data (because it may be kept in the shared cache). Furthermore, because the cost of bringing data into cache from shared memory is relatively high, it is best to try to make sure that all required references to a data item by all processors occurs while the data is in cache. While this may seem obvious, it has strong implications about the way algorithms are organized. In fact, Jalby, Meyer, and Gallivan have shown that a block structured algorithms achieve the best performance on the machine. Based on their results we (Jalby and Gannon) designed an FFT library for the Alliant that is very fast and we are now incorporating the block structuring transformations into the programming tools system.

On the Butterfly there is no shared cache and no strong need to do blocking. However, there is a related problem and solution. The memory on the butterfly is local. This means that when data is in a local memory the access is much faster than if it is far away. We discovered that the same analysis that was needed to keep data in cache for the Alliant could be used to decide which

data must be kept in the local memory of each Butterfly processor. This was a rather striking discovery which has led to uniform model of cache management described in the attached paper.

Artificial Intelligence, Production Systems and OPS 5.

Two other algorithm application areas that we hare looking at are related to Artificial Intelligence and Expert Systems. One is Neural Network Modeling which will not be described here and the other is parallelism experiments with the production system language OPS5. Production systems are used in the inference engines of expert systems. One of the most common is OPS5 and it is based on a tree resolution method called the Rete Match algorithm. We have now completed one implementation based on using butterfly Lisp on the BBN system. This proved to be far too slow partially due to compiler problems, but mostly due to the fact that the obvious ways to try to use concurrency in the match algorithm do not work. (This fact has been reported by several people in the literature).

We have started a new effort that will focus more energy on the lower levels of the computation that should prove to be effective for both the Alliant system and the Butterfly. We will report on the final results of this study in our final report.

Genetic Algorithms.

This work is being done by graduate student J. Y. Suh under the direction of faculty member Dirk Van Gucht. Genetic algorithms are an optimization technique that uses simple ideas from evolution theory to solve optimization problems. In this exercise we started with a good serial C program for doing a genetic optimization of the traveling salesman problem. We then did a mechanical set of transformations to come up with a reasonably good Butterfly version. A series of test were made which showed moderate performance improvements that resulted in speed-us of about 12 on a 16 processor machine.

By looking at where the restructured algorithm failed to perform with perfect speed-up he noticed that the serial algorithm was bound by a centralized control mechanism that inhibited parallelism. By focusing on this problem was able to design an completely new "distributed" genetic code. The new code has been run on Butterfly systems with as many as 128 processors with speed-ups of over 120.

3. FUTURE WORK

We feel that our attempts to help automate the process of restructuring serial program are going very well and that these tools that we are building are essential if we have any hope of solving the portablility problem. However, one clear message that has emerged from our experimental work. It is not possible to derive the OPTIMAL algorithms for any given computation by a purely mechanical set of transformations to the source code of a good serial algorithm. Algorithm RETHINKING is needed to do that. The important question to ask is what sort of tools are needed to help programmers with this process. It is our conclusion that the direction that we need to take this work is to find way to

help programmers with the process of redesigning algorithms.

Again, it is our experimental work that has led us to a way to solve this problem. The process that is usually followed by programmers in finding a new algorithm is to try to discover exactly why the old one failed. He does this by testing the program and isolating the serial bottlenecks in the computation and understanding why they take the form that they do. It is this process where programmer need the most help.

Our next set of tools will be built to help users identify serial bottleneck in algorithms. We will do this by building a a performance estimation "took kit" based on our research on estimating speed-up and memory hierarchy (cache and local memory) modeling theory. A full report on some initial experiments with this project will appear in the final report for this project.

END

DATE

FILMED

4-88

DTIC